The Lou data analysis nodes (LDANs) provide dedicated PBS resources to perform post-processing tasks on your Lou mass storage data. You can also use them for performing pre-processing or post-processing tasks on active Pleiades, Aitken, or Electra data. The LDANs can be accessed only through PBS jobs in which resources are dedicated to the job owner.

The LDANs contain Intel Xeon Gold 6154 "Skylake" processors, equipped with more memory than the Electra Skylake nodes, as follows:

- Each ldan[*11-12*] has 36 cores and 768 GB of memory
- Each ldan[*13-14*] has 36 cores and 1.5 TB of memory

The Lou, Lustre, and Pleiades home filesystems are mounted on the LDANs. You can access them in a PBS job running on an LDAN by using the following paths:

Lou:
  /u/*username*
Lustre:
  /nobackup/*username*
Pleiades:
  /pleiades/u/*username*

Software Modules

All of the software packages that you use on the Pleiades front-end (PFE) nodes or Pleiades/Aitken/Electra compute nodes are also available on the LDANs. However, both the Lou front-end (LFE) nodes and LDANs use startup files from your Lou home directory rather than your Pleiades home directory.

Therefore, if your PBS jobs rely on loading specific software modules or environment variables in your Pleiades system startup files (such as .cshrc), in order to run them on the LDANs you must modify your startup files on Lou to load the modules and set appropriate environment variables.

Running PBS Jobs on LDANs

The PBS server for the LDANs is pbspl1. To use the LDANs, submit your jobs to the ldan queue. Each job can use only *one* LDAN for up to *three days*, and each user can have a maximum of *two jobs* running simultaneously.

Before You Begin: If you want to process archive data that has been migrated offline to tape, use the Data Migration Facility command dmget to migrate the data back to disk before submitting jobs to the LDANs.

Submitting Your PBS Job

You can submit interactive PBS jobs to the LDANs from either the LFEs or the PFEs. You can submit PBS job scripts from either your Lustre home filesystem (/nobackup/*username)* or your Lou home filesystem (/u/*username)*.

WARNING: Do not submit job scripts from the Pleiades home filesystem, because PBS error and output files cannot be copied back there from the LDANs.

Use the :mem=*xxx*GB attribute to specify how much memory you need for your job. This will instruct PBS to allocate an appropriate LDAN.

Note: Keep in mind that the amount of memory on a node that is available for your job is slightly less than the total physical memory, because the system kernel can use up to 4 GB of memory in each node.

Sample PBS Script

```
#PBS -lselect=1:ncpus=36:mem=750GB
#PBS -lwalltime=2:00:00
#PBS -q ldan

### Remember to load necessary modules
### or set environment variables that you need

module load xxx

### Note that the default directory a PBS job
### is in when it starts is the Lou home filesystem.
### Change directory to the appropriate directory
### for your pre- or post- processing work.

cd $PBS_O_WORKDIR

### If your executable for pre- or post- processing
### is located under your Pleiades home filesystem,
### use the pathname /pleiades/u/username/bin/....
### do not use the pathname /u/username/bin/...

/pleiades/u/username/bin/your_processing_code < input > output
```

Running Graphics Applications on LDANs

If you want to run graphics applications that consume enough memory to require one of the larger-memory LDANs, submit an interactive PBS job from one of the LFEs that specifies mem=1040GB. For example:

```
lfe% qsub -I -q ldan -lselect=1:mem=1040GB,walltime=1:00:00
```

This will provide you with an LDAN for the duration of the specified wall time. Then, from any other terminal window, you can use SSH to connect directly to the LDAN assigned to your job, and run your graphics application interactively. If you have SSH passthrough set up, you can even SSH directly in to the LDAN from your workstation by adding the LDANs as allowed SSH passthrough targets in your .ssh/config file.

To run graphics applications directly from a PBS session submitted from an LFE, you need to export your DISPLAY environment by adding the -X option to the qsub command, as follows:

```
lfe% qsub -I -X -q ldan -lselect=1:mem=750GB,walltime=1:00:00
```

If you are a remote user, you can also explore using Virtual Network Computing (VNC) to connect to NAS systems.

The PFEs are the front-end systems for Pleiades, Aitken, Electra, and Endeavour. They provide an environment that enables quick turnaround for tasks such as file editing, file transferring, compiling, and short debugging/testing sessions, as well as for batch job submissions via PBS to a subset of the Pleiades compute nodes, or to Aitken, Electra, to Endeavour.

WARNING: The PFEs use Intel Sandy Bridge processors. If you use a PGI compiler to build your executable on the PFEs, be aware that by default the executable is optimized for the Sandy Bridge microarchitecture (which includes Sandy Bridge and Ivy Bridge) and will not necessarily execute on the compute node processors. See PGI Compilers and Tools for information on generating a single executable that will work on all processor types.

You cannot use SSH to access the compute nodes except for the subset of nodes your PBS job is running on.

Pre-Processing and Post-Processing Data

For pre- and post-processing applications such as Tecplot, IDL, and MATLAB, we recommend using the Lou data analysis nodes (LDANs). You can request the LDANs in the qsub command line. The Lou home filesystems, Pleiades home filesystems, and /nobackup filesystems are all mounted on the LDANs.

Restrictions on Front-End Systems

MPI (Message Passing Interface) jobs are *not* permitted to run on the PFEs. In addition, you will be notified by email if a job running on a PFE exceeds 27 GB.

Before starting a large-memory session, it is a good idea to make sure there is enough memory available. You can run the top command, hit "M", and check under the "RES" column for other large memory applications that may be running.

File Transfers to Mass Storage

The /nobackup filesystems are mounted on Lou, so the easiest way to transfer files between Pleiades and Lou is to initiate a command such as shiftc, cp, mcp, or tar on Lou. For example:

```
lou% shiftc /nobackup/username/filename $HOME
```

If you initiate the transfer from Pleiades, you can use the commands scp or shiftc to transfer files between a PFE and Lou. For very large file transfers, we recommend the NAS-developed Shift tool (shiftc).

File transfers from the compute nodes to Lou must first go through one of the PFEs.

When sending data to Lou, keep your largest individual file size under 1 terabyte (TB). Files larger than 1 TB will occupy all of the tape drives, preventing other file restores and backups.

Licensed Application Software

Licensed Application Software: Overview

There are a few licensed applications available for your use on NAS systems. You can find these in the /nasa directory.

The available licenses were either purchased by NAS—with the justification that many users need the particular applications—or by users themselves. If you would like to use a licensed application that is not yet available on NAS systems, you may need to purchase the license yourself.

Interactive Data Language (IDL)

Interactive Data Language (IDL) is a programming language for data analysis, visualization, and cross-platform application development. IDL combines tools for many types of projects, from "quick-look," interactive analysis and display to large-scale commercial programming projects.

NAS currently has IDL development licenses that allow 10 users to use the tool at the same time. Each license allows a single user to run multiple IDL sessions with the same session display simultaneously. Follow these steps to take advantage of this feature:

1. Use SSH to connect to a PFE.
2. Check the DISPLAY variable of the PFE, and launch multiple xterm sessions to run in the background. For example:

   ```
   pfe25% echo $DISPLAY
   pfe25:83.0

   pfe25% xterm&
   pfe25% xterm&
   pfe25% xterm&
   ```

3. In each xterm window, check the DISPLAY variable to confirm that it has the same number that was shown in the original window. Load an IDL module, then launch the idl command. For example:

   ```
   pfe25% echo $DISPLAY
   pfe25:83.0
   pfe25% module load idl/8.1
   pfe25% idl
   IDL>
   ```

In the example above, a single user occupies only one IDL license rather than three licenses by ensuring that the IDL sessions are running in three separate xterm windows that all have the same DISPLAY number.

Use one of the following methods to view how many licenses are currently in use:

- Run /u/scicon/tools/bin/check_licenses -i
- Load an IDL module and run the lmstat -a command

WARNING: You must quit IDL and release the license when you are done. Keeping the license in use when you are not actively using IDL prevents others who need access to IDL from doing useful work.

If you cannot use IDL because all licenses are currently being used, try again at a later time. If you notice that multiple licenses are occupied by the same user, please contact the NAS Control Room at (800) 331-8737, (650) 604-4444, or by email at support@nas.nasa.gov.

For more information, see:

- NAS IDL License Policy
- IDL Documentation
- IDL Programming Language (Wikipedia)

MATLAB

MATLAB is a numerical computing environment and programming language, created by MathWorks, that enables you to easily manipulate matrices, plot functions and data, implement algorithms, create user interfaces, and interface with programs in other languages. Although MATLAB specializes in numerical computing, an optional toolbox interfaces with the Maple symbolic engine, allowing it to be part of a full computer algebra system.

A total of 16 licenses are available to use on either NAS HPC systems or NAS-supported desktop systems. To find out how many licenses are currently in use, run:

```
/u/scicon/tools/bin/check_licenses -m (view general licenses only)

/u/scicon/tools/bin/check_licenses -M (view general licenses and compiler/toolboxes)
```

Then, load a MATLAB module:

```
% module load matlab/2016b
```

See the following websites for more information about MATLAB software:

- MATLAB documentation (MathWorks)
- MATLAB (Wikipedia)

See the following Knowledge Base articles for more information about NAS MATLAB licenses:

- MATLAB License Policy
- Using NAS MATLAB Licenses

Two Tecplot products are available on NAS systems: Tecplot 360 and Tecplot Chorus.

Tecplot 360

Tecplot 360 is a computational fluid dynamics and numerical simulation visualization software program used for post-processing simulation results. Common tasks associated with post-processing analysis of flow solvers (for example, Fluent, STAR-CD, OpenFOAM) include:

- Calculating grid quantities (such as aspect ratios, skewness, orthogonality, and stretch factors)
- Normalizing data; deriving flow field functions (such as pressure coefficient or vorticity magnitude)
- Verifying solution convergence
- Estimating the order of accuracy of solutions
- Interactively exploring data through cut planes (slices through a region), iso-surfaces (3D maps of concentrations), particle paths (dropping an object in the "fluid" and watching where it goes)

The NAS Tecplot 360 license does not restrict the number of instances of Tecplot 360 that can be run concurrently.

TIP: If you are having problems running Tecplot through VNC, you can try running it with the -mesa option, which uses software rendering.

Tecplot Chorus

Tecplot Chorus is a simulation analytics framework that unites physics visualization with data management and analytics in a single environment.

Currently, one Tecplot Chorus license is available to use on Pleiades or NAS-supported desktop systems.

For more information, see:

- Tecplot product documentation
- Tecplot (Wikipedia)

Modules and Packages

Software on NAS Systems

Software programs on NAS systems are managed as modules or packages. Available programs are listed in tables below.

Note: The name of a software module or package may contain additional information, such as the vendor name, version number, or what compiler/library is used to build the software. For example:

- comp-intel/2020.4.304 - Intel Compiler version 2020.4.304
- mpi-hpe/mpt.2.25 - HPE MPI library version 2.25
- netcdf/4.4.1.1_mpt - NetCDF version 4.4.1.1, built with HPE MPT

Modules

Use the `module avail` command to see all available software modules. Run `module whatis` to view a short description of every module. For more information about a specific module, run `module help` *modulename*.

See Using Software Modules for more information.

Available Modules (as of December 20, 2021)

Module	**Function**
boost	C++ library
cdo	Climate and NWP data analysis tool
comp-aocc	AMD Optimizing C/C++ Compiler
comp-intel	Intel Compiler
cuda	NVIDIA parallel computing platform and programming model
dakota	Software toolkit from Sandia
fieldview	Data visualization & analysis tool
firefox	Web browser
gcc	C/C++ compiler
hdf4	I/O library & tools
hdf5	I/O library & tools
idl	Data visualization & analysis tool
mathematica	System & language for mathematical application
matlab	Numerical computing environment & programming language
metis	Graph, mesh, & hypergraph partitioning software
mpi-hpe	HPE MPI library
mpi-intel	Intel MPI library
ncl	NCAR Command Language
nco	NetCDF Operators
ncview	Visual browser for NetCDF files
netcdf	I/O library
nvhpc	NVIDIA HPC Software Development Kit
octave	Numerical computations language
openfoam	Open source CFD software
pagecache-management	Tool for limiting page cache usage
paraview	Data analysis and scientific visualization application
parmetis	Math/numerical library
petsc	Portable, Extensible Toolkit for Scientific Computation
pkgsrc	NetBSD Package Source collection of software packages
python3	Programming language (version 3)
scicon	Tools created by NAS Scicon team
savors	NAS in-house utility for synchronization and visualization of arbitrary streams
singularity	HPC container platform
szip	Compression library
tecplot	Data visualization & analysis tool
totalview	Debugger
vtune	Performance analysis tool
x2go	Open source remote desktop software for Linux
xxdiff	Graphical file & directories comparison & merge tool

Packages in the NetBSD Package Source Collection (pkgsrc)

Many software programs are now managed as packages on NAS systems, via the NetBSD Package Source Collection. To use a package, you must first access the current package collection by loading the pkgsrc module, which is listed in the Modules table.

See Using Software in the NetBSD Packages Collection for more information.

Partial list of NetBSD Packages in pgksrc/2021Q2

Module	Function
bioperl	Perl tools for computational molecular biology
bison	GNU yacc(1) replacement
blas	Basic Linear Algebra System
bmake	Portable (autoconf) bersion of NetBSD "make" utility
bzr	Bazaar open source distributed version control system
clang	C language family front-end for LLVM
cscope	Interactive C program browser
fetch	Client to fetch URLs
ffmpeg4	Decoding, encoding, & streaming software (v4.x)
fftw	Fast C routines to compute DFTs
gdal	Translator library for raster geospacial data formats
gettext	Tools for providing messages in different languages
git	GIT version control suite meta-package
GMT	Generic Mapping Tools
gnuplot	Portable, interactive function-plotting utility
grace	GRaphing, Advanced Computation & Exploration of data
GraphicsMagick	X application for displaying & manipulating images
h5utils	Utilities for conversion to/from HDF5
hdf5-c++	New generation HDF C++ wrappers
htop	Enhanced version of "top" utility
ImageMagick	Package for display & interactive image manipulation
lrzip	Long range ZIP or Lzma RZIP
lz4	Extremely fast compression algorithm
lzop	Fast file compressor similar to gzip, using the LZO library
mercurial	Fast, lightweight source control management system
metis	Unstructured graph partitioning & sparse matrix ordering system
mg	Small, fast, public domain Emacs-style editor
mono	Open-source implementation of .NET Development Framework
nano	Small, friendly text editor (a replacement for Pico)
ncview	Visual browser for netCDF format files
netcdf-fortran	Fortran support for NetCDF
nvi	Berkeley nvi with additional features
pbzip2	Parallel implementation of bzip2 block-sorting file compressor
pkgfind	Find packages by package name in pkgsrc
qgis	Geographic information system (GIS)
R	Statistical language for data analysis & graphics
rhash	Calculate/check CRC32, MD5, SHA1, GOST, or other hash sums
ruby	Ruby programming language
tkcvs	Tcl/Tk front-ends to CVS & diff
tmux	BSD-licensed terminal multiplexer (GNU screen alternative)
valgrind	Debugging & profiling tools
vtk	Visualization toolkit
wget	Retrieve files from the internet via HTTP & FTP

Using Software Packages in pkgsrc

The NetBSD Packages Collection (pkgsrc) is a third-party software package management system for Unix-like operating systems. Among the over 17,000 packages available in pkgsrc, a few hundred of them have been built on Pleiades and are available in the public directory /nasa/pkgsrc.

A new collection of software packages is released every quarter by the pkgsrc developers. In addition to new packages or updated versions, each release is largely comprised of the same packages (with identical version numbers) as previous releases.

Software Packages Available on NAS Systems

For a list of software programs that are currently available as packages, see Available Software Modules and Packages.

You can also see which pkgsrc modules are currently available by running module avail pkgsrc. For example:

```
pkgsrc/2020Q4  pkgsrc/2021Q2
```

are available on systems running TOSS 3, and

```
pkgsrc/2022Q1-rome
```

is currently the only one available on the Aitken Rome nodes running TOSS 4.

Note: On Endeavour3/4, which still uses the SUSE Linux Enterprise Server Version 12 (SLES 12) operating system, be aware that the only pkgsrc modules available are older than 2021.

Note: Although the discussion and examples below use pkgsrc/2021Q2 and the path /nasa/pkgsrc/toss3/2021Q2, you should use pkgsrc/2022Q1-rome and the path /nasa/pkgsrc/toss4/2022Q1-rome when using the Aitken Rome nodes.

All of the source distributions for building the pkgsrc modules are stored in the /nasa/pkgsrc/distfiles directory. If you plan to build software from source, check to see whether the source distribution is already there before downloading the files from the internet.

To view a complete list of all packages and a one-line description of each package, load a pkgsrc module and type the pkg_info command without any additional options. If you are looking for a particular package, you can check whether it is available by using the grep command plus a keyword from the output of the pkg_info command.

TIP: Because some package names contain capital letters, add -i to the grep command to ignore case distinctions.

For example:

```
pfe% module load pkgsrc/2021Q2
pfe% pkg_info | grep -i cairo
cairo-1.16.0nb4         Vector graphics library with cross-device output support
cairo-gobject-1.16.0nb5 Vector graphics library with cross-device output support
py39-cairo-1.20.1       Python bindings for cairo
```

You can also use the -e option to test the existence of a given package. For example:

```
pfe% pkg_info -e cairo

pfe% cairo-1.16.0nb4
```

Once you know the package name, you can use additional pkg_info options to find more information about the package. You can specify the full package name, or you can leave out the version number. For example:

```
pfe% pkg_info -B cairo-1.16.0nb4
```

or

```
pfe% pkg_info -B cairo
```

Some useful options for pkg_info include:

-B
: shows build information of the package

-d
: shows a long description of the package

-L
: shows the files within the package

-N
: shows what other packages were used to build the package

-R
: shows what other packages require the package

Using Packages

The root directory of a pkgsrc release (for example, /nasa/pkgsrc/toss3/2021Q2) contains familiar directories such as bin, sbin, lib, include, man, and info. For most packages, you can find their executables, libraries, or include files in these directories.

Using an Executable

When you load a pkgsrc module, the paths to its bin and sbin directories are prepended to the PATH environment variable, as shown in the following example. Therefore, you do not need to provide full paths to these directories in order to run executables in them.

```
pfe% module show pkgsrc/2021Q2
-------------------------------------------------------------------
/nasa/modulefiles/pkgsrc/toss3/pkgsrc/2021Q2:

system           /bin/logger -p local2.info -t envmodule ychang1 display pkgsrc/2021Q2
module-whatis    A collection of software built via NetBSD's pkgsrc
prepend-path     INFOPATH /nasa/pkgsrc/toss3/2021Q2/info
prepend-path     MANPATH /nasa/pkgsrc/toss3/2021Q2/man
prepend-path     PATH /nasa/pkgsrc/toss3/2021Q2/bin:/nasa/pkgsrc/toss3/2021Q2/sbin
setenv           PKGSRC_BASE /nasa/pkgsrc/toss3/2021Q2
-------------------------------------------------------------------
```

Linking a Library

Unlike the PATH environment variable, loading a pkgsrc module does not set the LD_LIBRARY_PATH environment variable. The reason for this is to prevent libraries in the pkgsrc directory from overriding those with the same filenames at the system default locations or directories that may have been included in your existing LD_LIBRARY_PATH.

To link a static library (for example, libcairo.a) in the /nasa/pkgsrc/toss3/2021Q2/lib directory:

```
-I/nasa/pkgsrc/toss3/2021Q2/include
-L/nasa/pkgsrc/toss3/2021Q2/lib -lcairo
```

To link to a dynamic library (for example, libfftw.so) in the /nasa/pkgsrc/toss3/2021Q2/lib directory, follow the recommendations in the list below, in order of preference.

Note: Dynamic libraries have the suffix ".so" rather than the suffix ".a", which is used for static libraries.

- Set the LD_RUN_PATH environment variable as shown below prior to the link step. This will get /nasa/pkgsrc/toss3/2021Q2/lib encoded into the executable. Setting LD_RUN_PATH or LD_LIBRARY_PATH is not needed during run time.

  ```
  setenv LD_RUN_PATH /nasa/pkgsrc/toss3/2021Q2/lib
  -I/nasa/pkgsrc/toss3/2021Q2/include
  -L/nasa/pkgsrc/toss3/2021Q2/lib -lfftw
  ```

- Add the -Wl,-R linker option during the link step. This will also get /nasa/pkgsrc/toss3/2021Q2/lib encoded into the executable. Setting LD_RUN_PATH or LD_LIBRARY_PATH is not needed during run time.

  Note: The linker option -Wl,-R takes precedence over the LD_RUN_PATH setting.

  ```
  -I/nasa/pkgsrc/toss3/2021Q2/include
  -L/nasa/pkgsrc/toss3/2021Q2/lib -lfftw
  -Wl,-R/nasa/pkgsrc/toss3/2021Q2/lib
  ```

- Do not set LD_RUN_PATH or use the -Wl,-R linker option during the link step. Since /nasa/pkgsrc/toss3/2021Q2/lib is not encoded into the executable, you must set LD_LIBRARY_PATH during run time.

  Link time:

  ```
  -I/nasa/pkgsrc/toss3/2021Q2/include
  -L/nasa/pkgsrc/toss3/2021Q2/lib -lfftw
  ```

  Run time:

  ```
  setenv LD_LIBRARY_PATH "/nasa/pkgsrc/toss3/2021Q2/lib:$LD_LIBRARY_PATH"
  ```

For more information about using the setenv command, see **man setenv**.

The files of some packages are not installed in the bin, lib, and include directories. For example, the guile package is installed in the /nasa/pkgsrc/toss3/2021Q2/guile directory, which has its own bin, lib, and include subdirectories. The best way to find the exact location of a file you need from a package is to run the pkg_info command as follows:

```
pfe% pkg_info -L package_name
```

Separate modules have been created for some commonly used packages to make them easier to use—you can simply load the modulefile without loading a pkgsrc module. On systems running TOSS 3, these include:

- boost/1.76
- gcc/7.5
- gcc/9.3
- gcc/10.2
- gcc/10.3
- python3/3.9.5

- octave/6.2

On the Aitken Rome nodes running TOSS 4, the following modules from pkgsrc have been made available:

- boost/1.78
- gcc/6.5
- gcc/7.5
- gcc/8.4
- gcc/9.3
- gcc/10.3
- python3/3.9.12
- octave/6.4

Note: /usr/bin/python will no longer be linked to python2. To use python2, you must specify the whole path /usr/bin/python2.

Separate modules for other packages can be created when there is demand from the NAS user community.

For more information about pkgsrc, see https://www.pkgsrc.org.

Using Software Modules

See the commands described below to learn how to use modules.

Note: Software programs on NAS systems are managed either as modules or packages. For information about using software programs that are managed as packages, see Using Software in the NetBSD Packages Collection.

Using Modules

To find out what software modules are available on NAS systems, run the `module avail` command. You can also view a short description of every module by running `module whatis`, or find information about a specific module by running `module help` *modulename*.

To use the software provided in a module, you must first load the module by running the `module load` command. You can load several modules at the same time, as follows:

```
% module load module_name_1 module_name_2 ... module_name_N
```

Testing Modules

New software modules and new versions of existing modules are sometimes available for testing before they are released for general use. To access the test modules, run:

```
module use -a /nasa/modulefiles/testing
module avail
module load module_name
```

If you use the test modules, please send your feedback to support@nas.nasa.gov. Let us know whether the modules worked well or, if you had any problems, describe any steps you took to reproduce them.

Note: Test modules may be moved into production or deleted without notice.

Additional Module Commands

Some useful module commands include:

`module list`
: Lists which modules are already loaded in your environment.

`module purge`
: Removes all the modules that are loaded in your environment.

`module switch` *old_module_name new_module_name*
: Enables you to switch between two modules.

`module show` *module_name*
: Shows changes that will occur in your environment if you load *module_name.*

For more information, see **man module**.

TIP: If the module commands don't work, ensure that the following line
is included in your .cshrc or .profile file:

- For csh: `source /usr/local/lib/global.cshrc`
- For bash: `source /usr/local/lib/global.profile`

Creating Your Own Modulefiles

If you maintain software for your own use or to share with your group, it may be useful to create modulefiles that can be loaded, purged, and so on, just like any other modulefile.

To create your own modulefile, you can simply copy an existing modulefile to your directory (commonly named /u/*your_username*/privatemodules), and edit it as appropriate.

First, find a module that you want to mimic and run `module show` *module_name*. The modulefile itself will be listed at the top of the `module show` output. You can find most available modulefiles in the /nasa/modulefiles/toss3 directory.

Here are some common features in well-designed modulefiles:

```
#%Module
##
##  All modulefiles must begin on the first line with those 8 characters
##  Here, in the comments, you can describe how the package was built, or, better yet,
##  put it in the ModulesHelp section below

proc ModulesHelp { } {
   puts stderr "Put the information you want to display when the user
   does module help on this"
   puts stderr "modulefile. Note each continuation line begins with puts
   stderr and comments in double quotes"
}

# include the following line only if you want to log the loading of the
# modulefile in our system logs
# most users would leave this out
source /nasa/modulefiles/common/log_module_usage.tcl

# if there might be several versions of this software, it would be good to
#set the version number
set version 1.0-something

# setting basepath or SWDIR is for naming the root directory where your
# various bin, include, lib directories reside
# see later in the modulefile on how it is used
set basepath /u/your_userid/dir_name/dir_name

# if this software requires other modules to be loaded first,
# then use the prereq specifier, for example:
prereq comp-intel

# if your software requires other software that you build,
#then you might simply load them, as in:
module load  my_other_module1  my_other_module2

# if this software conflicts with another software that the user
# may inadvertently load, then use:
conflict another_modulefile

# Finally, the actual setting of the PATHs etc.  You have a choice of either
# prepend-path or append-path:
prepend-path   PATH                      $basepath/$version/bin
prepend-path   CPATH                     $basepath/$version/include
prepend-path   FPATH                     $basepath/$version/include
prepend-path   LD_LIBRARY_PATH           $basepath/$version/lib
prepend-path   LIBRARY_PATH              $basepath/$version/lib
prepend-path   MANPATH                   $basepath/$version/share/man
```

If you plan to share this software with other users in your group, make sure that they have read permission to every file and directory, and execute permission on all directories and the executables in the software's bin directory.

In order to see your newly created modulefile in the output of the module avail command, run the module use command first, as follows:

```
% module use -a /u/your_username/privatemodules
```

Note: The -a option puts the new modulefile at the end of module avail output. Without it, the modulefile will be listed at the top.